intel®

APPLICATION NOTE

AP-179

October 1986

PROM Programming with the Intel Personal Development System (iPDS™)

FRED MOSEDALE
DSHO TECHNICAL PUBLICATIONS

Order Number: 280015-001

- **Need for simple operation**—You want a programming device that satisfies all of the following needs and is simple to operate. You do not want to have to refer to a manual every time you wish to program a PROM.
- **Need to program a wide variety of PROMs**—For greatest flexibility, you want a programming device that can program the various kinds of PROMs that are available. For example, you will want to be able to program microcontrollers with EPROMs, and you will want to be able to program from the small inexpensive 16K and 32K PROMS to the latest 256K PROMS with int$_e$ligent Programming™ algorithms to speed programming. You will also want to be able to program those PROMs that use the new lower programming voltage (12.5V).
- **Need to be upgradeable**—A PROM programming device should be designed so that it can be upgraded to program PROMs that will be available in the future. Without upgradeability, the device will soon be out-of-date.
- **Need to check PROM contents before programming**—If the PROM you will be programming is blank, it can of course be programmed. Even if it has some bits set at the time of programming, if the same bits must also be set for the program, the PROM can be used. Thus, ideally, a PROM programming device will determine whether the PROM is blank, and if not, determine whether the bits already set are compatible with the program to be loaded into the PROM.
- **Need to recognize file formats**—When a PROM is programmed using an object file generated by a compiler or assembler, the PROM programming device must be able to extract the data that is to be loaded into the PROM from the larger file structure. For greatest flexibility, you will need a PROM programming device that recognizes the file structures generated by compilers and assemblers that you will use when developing program code for the PROMs.
- **Need to support a variety of source program options**—There are three sources you may wish to use for PROM data: an already-programmed PROM, a software development system, or a disk. For greatest programming flexibility, you will want PROM programming device that makes all three kinds of sources available.
- **Need to support data manipulation and modification**—You may wish to modify a source file for your PROM program. For example, you may discover an error in the source file, or you may realize that for your new processor system, the PROM data must first be 2's complemented. For a variety of reasons, your PROM programming will be much more flexible if the PROM programming device offers a buffer for temporary storage and PROM programming software that can manipulate the source program data in a variety of ways.
- **Need for variety in loading program code into PROMs**—If your product has a 16-bit microprocessor and you are using 8-bit PROMs to store the firmware, you will need to interleave the 16-bit code between two 8-bit PROMs. A PROM programming device with interleaving capability will speed such programming. You may want other kinds of flexibility when programming.
- **Need to transfer long programs that will not fit into one PROM**—Long programs may exceed the storage capacity of the PROMs chosen for your product. You need a programming device that can format the program so that it can be stored in successive PROMs.
- **Need to verify programming**—When programming is finished, you will want to check that the PROM is indeed correctly programmed. A defect in the PROM could corrupt the intended firmware. Checking would involve comparing the source with the programmed PROM.
- **Need to compare buffer with PROM**—If you are interrupted when programming PROMs or if you have not labeled PROMs that you did program, you may forget whether a particular PROM was programmed. In such cases, you will want to compare the particular PROM with the program stored in the buffer of the programming device. Comparison will prevent you from having to reprogram an already-programmed PROM.

INTRODUCTION

Programmable read-only memory (PROM) devices play a significant role in microprocessor-based products. How can PROM programming devices perform to best serve the needs of those who develop and service such products?

This application note first provides a general answer to this question; then, it proceeds to describe the features and use of PROM programming hardware and software available for the Intel Personal Development System (iPDS™). The description explains how the iPDS system provides the wide range of capabilities needed by those who use PROM programming devices. The description also highlights the iPDS system's ability to program some of Intel's newest EPROMs.

PROMS IN THE LIFE CYCLE OF A PRODUCT

Memory Options: A Review

Before PROM programming needs are discussed, it is important to briefly review memory options available to designers.

Microprocessor-based products need memory to store instructions and data used in controlling their operations. In order to maximize product operating speeds, designers must use memory that can be accessed quickly. Both random access memory (RAM) devices and read-only memory (ROM) devices offer designers quick access, but RAM devices are volatile—their contents are erased when system power is turned off. ROM devices are nonvolatile; thus designers use ROMs to store programs and data that will not change during the operation of the product.

After a product's program code data has been debugged, you can transfer the code to programmable ROMs (PROMs) or masked ROMs. (The most flexible PROMs are E^2PROMs and EPROMs; E^2PROMs are electrically erasable and EPROMs can be erased by ultraviolet light.) PROMs are programmed using a relatively simple procedure; by contrast, masked ROMs can only be programmed in a manufacturing environment. Thus, masked ROMs provide less flexibility but are used because they may be more cost-effective in large volumes. However, because the price of PROMs is falling and because inventories with erasable PROMs can be reprogrammed when product programs are changed, erasable PROMs are also attractive for large volumes.

Desirable PROM Programming Features

Once your product's software is debugged, you can load the software into the product's PROMs (so that it becomes the product's firmware). However, usually in the development of a product, the initial programming of PROM devices is not the last operation involving the PROMs. Even if the software is debugged, once it is loaded into the PROMs, you may discover new bugs in the program that you failed to detect before the program was committed to PROMs. So, there may soon be a need to erase the PROMs (if they are EPROMs or E^2PROMs) and reprogram them.

During product development and servicing, you will also sometimes need to accomplish the following tasks:

- Check the contents of a PROM.
- Use one PROM to program other PROMs that will be used in other prototype systems.
- Update earlier firmware versions with later versions.

Consider in more detail the (P)ROM-related needs that can arise for you during the product's life cycle, that is, between the time when the product's software has been loaded into PROMs and the time when the product is phased out. There are two basic sets of needs, those having to do with displaying (P)ROM contents and those having to do with programming PROMS.

DISPLAYING AND PRINTING NEEDS

For a variety of reasons, you may need to examine what is stored in a (P)ROM. For example, you may suspect that a (P)ROM's program is in error; or you may have incomplete documentation on what was programmed into a (P)ROM. Thus, you will want to be able to display the contents on a video display terminal and to have a printer print out the contents. You will want to be able to choose the display base (binary, octal, decimal, or hexadecimal) and whether to display the contents as ASCII characters.

PROGRAMMING NEEDS

When you program PROMs, you need a programming device that can program a vareity of PROMS and one that offers flexibility and ease of programming. The following list describes PROM programming needs.

- **Need to lock microcontrollers from unauthorized access**—Some advanced microcontrollers can be locked to prevent unauthorized access. To take advantage of this security features, you need to be able to control the locking of the microcontrollers.
- **Need to automate routine PROM programming tasks**—To speed programming, you will want a programming device that can automate routine programming functions. Automation will not only speed programming, it will also release personnel for other work.

DESIRABLE PROM PROGRAMMING FEATURES: A SUMMARY

In summary, if PROMs (and ROMs) are incorporated in your product, you will have the greatest flexibility if your PROM programming device has the following features. (Of course, for ROMs only reading tasks are needed.)

- Is easy-to-use.
- Can program a wide variety of PROMs.
- Is upgradeable.
- Can display (P)ROM contents in ASCII characters and a variety of bases.
- Can enable a printer to print out (P)ROM contents in ASCII characters and in a variety of bases.
- Can check for blank PROMs.
- If PROM is not blank, can check PROM contents for compatibility with program.
- Can recognize file formats of your development system object files.
- Supports transfers of program code from development systems, disks, and other PROMs to the new PROM.
- Provides temporary storage and software for manipulating Programming data before loading it into the PROM.
- Supports variety in how data is loaded into PROMs, e.g.:
  - Interleaving 16-bit data into 8-bit PROMs.
  - Segmenting long programs so that resulting program segments fit into successive PROMs.
- Can verify the accuracy of copying.
- Can compare programming buffer with PROM contents.
- Can lock microcontrollers from unauthorized access.
- Can automate routine PROM programming tasks.

The Intel Personal Development system (iPDS) with the PROM programming option meets these needs. The following sections describe the iPDS PROM programming hardware and software and show how this system can perform all of these tasks for a variety of PROMs.

THE iPDS™ PROM PROGRAMMING SYSTEM

The iPDS system supports integrated hardware and software development; it provides a complete set of software development tools and in-circuit emulators for hardware debugging and hardware-software integration. With its optional PROM programming hardware and software, the iPDS system also supports PROM programming.

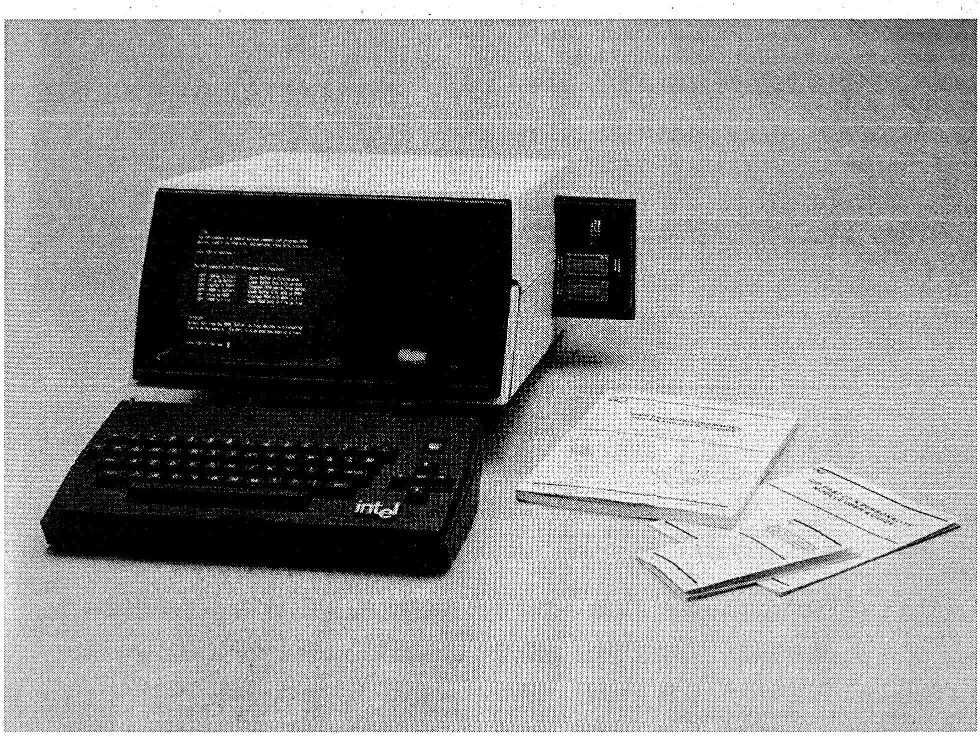
Three components comprise the iPDS PROM programming system: the iPDS system (with the ISIS-PDS operating system and the plug-in module adapter board), the PROM programming modules, and the prom programming software. Each of these components is described briefly in the following sections.

iPDS™ System

To perform PROM programming tasks, the iPDS system must use its ISIS-PDS operating system and the plug-in module adapter board. The PROM programming software (iPPS-PDS) runs under the ISIS-PDS operating system.

The adapter board allows you to use both the PROM programming personality modules and emulation modules. It provides the interface between the modules and the iPDS system.

Figure 1 shows the IPDS system with a PROM programming personality module plugged into its side.



280015-1

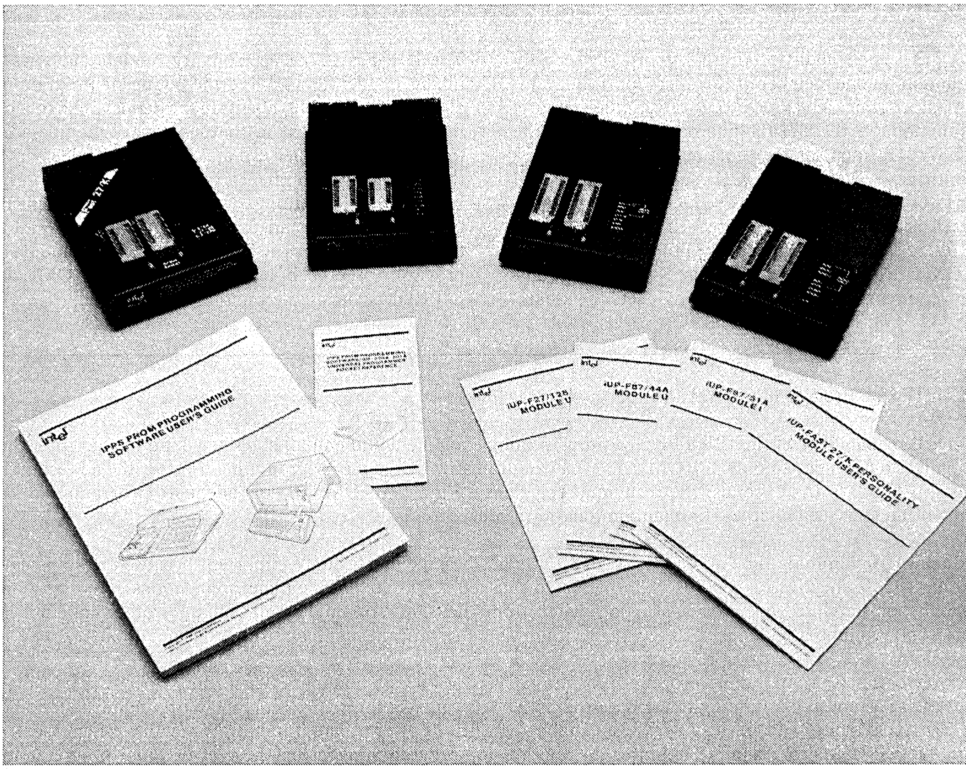
Figure 1. iPDS™ System with PROM Programming Personality Module

PROM Programming Personality Modules

A personality module is the interface between the iPDS system and a selected PROM. Personality modules contain all the hardware and firmware for reading and programming a family of Intel devices. Each personality module is a single molded unit inserted into the side panel of the iPDS unit. No additional adapters or sockets are needed. Table 1 lists the available personality modules, and Figure 2 shows the four modules.

Table 1. PROM Programming Personality Modules

Personality Module	PROM Type Programmed	PROMs and ROMs Supported
iUP-Fast 27/K	EPROM	2764, 2764A, 27128, 27256, and provisions for future PROMs.
iUP-F27/128	E^2/EPROM	2716, 2732, 2832A, 2764, 27128, 2815, and 2816
iUP-F87/51A	Microcontroller	8748, 8748H, 8048, 8749H, 8048H, 8049, 8049H, 8050H, 8751, 8751H, 8051
iUP-F87/44A	Peripheral	8741A, 8041A, 8742, 8042, 8744H, 8044AH, 8755A



280015-2

Figure 2. PROM Programming Personality Modules

Each personality module connects to the iPDS system through a 41-pin connector. Module firmware is uploaded into the iPDS system and executed by the IPDS system. The personality module firmware contains routines needed to read and program a family of PROMs. In addition, the personality module sends specific information about the selected PROM to the iPDS system, such as information about the PROM size and its blank state.

LEDs on each personality module indicate its operational status. On some personality modules a column of LEDs or a hexadecimal display indicates which PROM device type the user has selected. On some personality modules with more than one socket, an LED below each socket indicates the socket to be used. In addition, a red indicator light tells the user when power is being supplied to the selected device.

The personality module firmware performs selected PROM tests and indicates status:

- The PROM installation test verifies that the device is installed in the module correctly and that the ZIF socket is closed.
- The PROM blank check determines whether the device is blank. The iPDS system automatically determines whether the blank state for the particular device is defined as all zeros or all ones.
- The overlay check (performed when a PROM is not blank) determines which bits are programmed, compares those bits against the program to be loaded, and allows programming to continue if they match.

Easy-to-read status messages are also provided. The user can invoke all of the PROM device integrity checks except the installation test (which occurs automatically any time an operation is selected). The following sections describe specific features of the three personality modules that program the newer Intel PROMs.

IUP-F87/44A AND IUP-F87/51A PERSONALITY MODULES: SPECIAL FEATURE

Each of these personality modules supports the security bit function on one member of the microcontroller family it can program. The iUP-F87/44A module supports the function on the 8744H microcontroller, and the iUP-F87/51A supports the function on the 8751H microcontroller. The KEYLOCK command locks the 8744H (or the 8751H) EPROM memory from unauthorized access by setting the security bit; the microcontroller cannot be unlocked without erasing the EPROM. As a safety precaution, the KEYLOCK command requires user verification before it sets the security bit.

IUP-FAST 27/K PERSONALITY MODULE: SPECIAL FEATURES

The iUP-Fast 27/K personality module supports the int$_e$ligent Identifier™ and the int$_e$ligent Programming algorithms. The int$_e$ligent Identifier is used to check the PROM installed in the personality module socket to determine whether it matches the type selected; then the int$_e$ligent Identifier is used to select the proper int$_e$ligent Programming algorithms. The int$_e$ligent Programming Algorithms reduce PROM programming time by as much as a factor of ten. This module has provision for support of future EPROMs and E^2PROMs via simple plug-in updates.

The int$_e$ligent Programming™ Algorithm

Using the capabilities of the IPDS PROM programming equipment and employing a new kind of algorithm that recognizes differences among EPROM cells, you can dramatically reduce programming time for the newest high-density EPROMs. As a bonus, the technique helps ensure that EPROMs receive adequate programming—in terms of memory-cell charge—to maintain long-term reliability.

Reducing programming time and costs for EPROMs has become increasingly important because the chips have become a cost-effective, easy-to-use alternative to masked ROM in high-volume applications requiring code flexibility or simplified inventory—a major switch from EPROMs' original small-volume prototyping applications. And, volume usage makes EPROM programming a significant manufacturing consideration.

The conventional programming procedure for most EPROMs uses a nominal 30-msec pulse per EPROM byte, resulting in a total programming time of approximately 1.5 minutes for a 16K-bit chip. With the introduction of the 2764 (64K bits) and devices with even higher densities, however, programming times have increased. A 256K-bit EPROM, for example, requires 24 minutes for programming using the conventional programming method.

Most EPROM cells program in less than 45 msec, however. In fact, empirical data shows that very few cells require longer than 8 msec for programming. Therefore, a procedure that takes into account the characteristics of individual EPROM cells can significantly reduce a device's programming time.

Arbitrarily reducing programming time is risky, however, because a cell's ability to achieve and maintain its programmed state is a function of this time. What is needed, therefore, is a way to verify the level to which individual cells have been programmed. Such a way exists. By determining the charge stored in a cell compared to the minimum charge needed to program the cell to a detectable level, you can check for a program margin that ensures reliable EPROM operation.

Margin checking does not occur in conventional EPROM programming, however. Instead, each EPROM cell receives a 45- to 55-msec write pulse, and manufacturers attempts to ensure program margin by screening out EPROMs having bytes that do not program within 45 msec. This programming procedure is thus an open loop—no actual verification of margin occurs.

By contrast, the int$_e$ligent Progrmaming algorithm guarantees reliability through the closed-loop technique of margin checking. This algorithm uses two different pulse types: initial and over-program. The algorithm first applies a 1-msec initial pulse to an EPROM. After the pulse, it checks the EPROM's output for the desired programmed value. If the output is incorrect, the algorithm repeats the pulse-and-check operation. When the output is correct, the algorithm supplies an over-program pulse; the length of this pulse depends on how many initial pulses were used and varies with the EPROM being programmed. This longer pulse helps ensure that the EPROM cell has an adequate programming margin for reliable operation.

Prom Programming Software (iPPS—PDS)

The iPPS-PDS software provides easy-to-use commands that allow you to load programs into a target PROM from another PROM, from iPDS system memory, or directly from a disk file.

The iPPS-PDS software also supports data manipulation in the following Intel formats: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, and 286 absolute object. Addresses and data can be displayed in binary, octal, decimal, or hexadecimal. You can easily change default data format as well as number bases.

You invoke the iPPS-PDS software from the ISIS operating system. (The software can be run under control of ISIS submit files, thereby freeing you from repetitious command entry.)

An explanation of the iPPS-PDS software follows. It is divided into three main sections: the iPPS-PDS storage devices, iPPS-PDS commands, and invoking the iPPS-PDS. Also see the Appendix for iPDS PROM programming examples.

iPPS-PDS STORAGE DEVICES

The iPPS-PDS software transfers data between any two of the three storage devices: PROM, buffer, and file. These devices are defined in the following three sections.

PROM Device

The PROM device is plugged into a socket on the personality module installed in the iPDS system. The iPPS-PDS software does not recognize the PROM device until you enter the TYPE command. The TYPE command automatically sets the appropriate buffer size according to the size of the PROM device specified.

Buffer Device

The buffer device is a section of development system memory that the iPPS-PDS software allocates and uses as a working area for temporary storage and for rearranging data. Its boundaries can exist anywhere in a virtual address range from 0 to 16777215 (0 to $2^{24}-1$).

When the iPPS-PDS software is initialized, the buffer starting address is set to 0, and the buffer ending address is set to 8K − 1, providing an initial buffer size of 8K bytes (the default buffer size when no PROM type is specified). During subsequent iPPS-PDS operations, the size and boundaries can vary. Specific iPPS-PDS commands determine these variations. The most recent command that changed the lower boundary of the buffer determines the buffer starting address. The TYPE command affects both the size and location of the buffer. For example, the TYPE command always resets the buffer start address to 0. The most recent TYPE command controls the size of the buffer.

The iPDS system needs a virtual buffer when PROM size exceeds 8K. If the PROM size exceeds the 8K memory buffer space available on the development system, the iPPS-PDS software creates a virtual buffer area using temporary file space on disk.

Two temporary work files are used to create the virtual buffer. During subsequent virtual buffer operations, the iPPS-PDS software automatically swaps data in and out of development system memory from and to work files.

File Device

The file device is an ISIS file on a disk. It is specified within iPPS-PDS commands.

The data stored in the disk file is in one of the following Intel absolute formats: 8080 hexadecimal, 8080 object, 8086 hexadecimal, 8086 object, or 80286 object. The iPPS-PDS software can read any of these formats as input but writes data to a file in 8080 object, 8086 object, or 80286 object formats only. Basically, these files contain representations of blocks of memory data. Included with the data are addresses for the locations of the data. The data blocks are not necessarily in consecutive address order. The method used to create the file determines the order of the data.

The iPPS-PDS file device has address boundaries that exist in the virtual range from 0 to 16777215 (0 to $2^{24}-1$). These boundaries are determined as follows:

- The file's lowest address is the lowest address encountered while reading the file.
- The file's highest address is the highest address encountered while reading the file.

If the iPPS-PDS software creates the file (that is, if the file is a destination device in an iPPS-PDS command), the specific command issued determines these boundaries.

When you specify a particular address range to be read from a file, all sections in the address range that are not present in the file are written in a PROM destination device as the blank state of the currently selected PROM type. If the destination device is the buffer, the nonexistent sections in the file do not overwrite the corresponding sections in the buffer.

During the operation of commands that use the file device as a source, the iPPS-PDS software only reads the actual data from the file and ignores any other information in the file. For example, the file can contain special information used later for debugging. Since the iPPS-PDS software ignores this information, it will not appear in any new files generated. If the data is written back to the original file, the original file is deleted.

iPPS-PDS COMMANDS

Each iPPS-PDS command consists of a keyword that identifies the command, followed by other keywords and associated parameters that are the arguments of the command. You enter all iPPS-PDS commands, as well as program address and data information, through the development system ASCII keyboard; the commands are displayed on the system CRT. Table 2 summarizes the iPPS-PDS commands.

Table 2. iPPS-PDS Command Summary

Command	Description
Program Control Group	**Controls Execution of the iPPS-PDS Software.**
EXIT	Exits the iPPS software and returns control to the ISIS operating system.
<ESC>	Terminates the current command.
REPEAT	Repeats the previous command
ALTER	Edits and re-executes the previous command.
Utility Group	**Displays User Information and Status; Sets Default Values.**
DISPLAY	Displays PROM, buffer, or file data on the console.
PRINT	Prints PROM, buffer, or file data on the local printer.
HELP	Displays user assistance information.
MAP	Displays buffer structure and status.
BLANKCHECK	Checks for unprogrammed PROMs.
OVERLAY	Checks whether non-blank PROMs can be programmed.
TYPE	Selects the PROM type.
INITIALIZE	Initializes default number base and file type.
WORKFILES	Specifies the drive device for temporary work files.
Buffer Group	**Edits, Modifies, and Verifies Data in Buffer.**
SUBSTITUTE	Examines and modifies buffer data.
LOADDATA	Loads a section of buffer with a constant.
VERIFY	Verifies data in the PROM with buffer data.
Formatting Group	**Rearranges Data from PROM, Buffer, or File.**
Format	Formats and interleaves buffer, PROM, or file data.
Copy Group	**Copies Data from One Device to Another.**
COPY (file to PROM)	Programs PROM with data in a file on disk.
COPY (PROM to file)	Saves PROM data in a file on disk.
COPY (buffer to PROM)	Programs PROM with data from the buffer.
COPY (PROM to buffer)	Loads the buffer with data in the PROM.
COPY (buffer to file)	Saves the contents of buffer in a file on disk.
COPY (file to buffer)	Loads the buffer from a file on disk.
Security Group	**Locks Selected Devices; Prevents Unauthorized Access.**
KEYLOCK	Locks the PROM from unauthorized access.

Once entered, a command line is verified for correct syntax and executed. If a syntax error is detected, the following error message is dispalyed:

--SYNTAX ERROR--*specific error.*

If you omit a required keyword, the iPPS-PDS software prompts for the keyword and its associated parameters: If the keyword is entered but its parameters are omitted, either a default value is assumed or an error message is displayed if there is no default. In certain commands, default keywords are also assumed.

You can enter complete iPPS-PDS keywords or any unique abbreviation (only the first character is required). For example, command keywords of C, CO, COP, and COPY are all interpreted as the COPY command.

The iPPS-PDS software accepts numeric entries in any one of four number bases: binary (Y), octal (O or Q), decimal (T), or hexadecimal (H). Numbers can be entered in any of these bases by appending the appropriate letter identifier to specify the base (e.g., 11111111Y, 377Q, 255T, FFH). An explicit number base identifier overrides the default number base, which is initially hexadecimal.

INVOKING iPPS

There are two methods of invoking the iPPS-PDS software: command lines and submit files.

The command line for invoking the iPPS-PDS software (under V1.0 and later versions of the ISIS.PDS operating system) uses the following syntax:

[:F*n*:]IPPS

The symbol ":F*n*:" Specifies the drive on which the iPPS-PDS files are located. When you enter the iPPS-PDS command, the ISIS operating system loads and executes the iPPS-PDS software.

The iPPS-PDS software can also run under the control of a submit file. SUBMIT is an ISIS command that allows you to use a disk text file as input for further ISIS commands or as command inputs to utilities running under the ISIS operating system. Thus, a submit file can contain the ISIS command line to invoke the iPPS-PDS software and then a sequence of commands for the iPPS-PDS software itself.

Summary: the iPDS™ System Meets PROM Programming Needs

Table 3 describes briefly how the iPDS system meets each of the needs identified earlier in this application note.

The iPDS system can be a complete intelligent PROM programmer—and, because the iPDS system is also a development system, it can provide an excellent means to off-load PROM programming from your current development system (just as the iPDS system allows you to off-load other 8-bit development tasks). In addition, with its state-of-the-art PROM programming capability, the iPDS system becomes an attractive solution to your complete development system needs.

Table 3. iPDS™ Features Meet PROM Programming Needs

Need	iPDS™ Feature
Be easy-to-use.	iPPS software and the PROM programming personality modules were designed to provide ease-of-use.
Program a wide variety of PROMs.	Personality modules each permit the programming of a family of PROMs or microcontrollers.
Be upgradeable.	New personality modules will be released as new PROM families appear.
Display (P)ROM contents in ASCII characters or in a variety of bases.	iPPS DISPLAY command displays (P)ROM (or buffer or file) contents in ASCII characters and in binary, octal, decimal, or hexadecimal.
Enable a printer to print out (P)ROM contents in ASCII characters and in a variety of bases.	iPPS PRINT command prints out (P)ROM (or file or buffer) contents in ASCII characters and in binary, octal, decimal, or hexadecimal.
Check for blank PROMs.	iPPS BLANKCHECK command checks for blank PROMS.
If PROM is not blank, check PROM contents for compatibility with program.	iPPS OVERLAY command checks PROM contents for compatibility with program.
Recognize file formats of development system object files.	iPPS command file switch allows you to indicate to the iPDS system which object file format is being used.
Support transfers of program code from development system, disks, and other PROMs to the new PROM.	iPPS COPY commands allow you to copy in either direction between the iPDS disk drive(s), PROMs, and the iPDS buffer storage.
Provide temporary storage and software for manipulating programming data before loading it into the PROM.	iPDS buffer provides temporary storage and the iPPS SUBSTITUTE and LOADDATA commands allow you to manipulate programming data before you load it into a PROM.
Load data into PROMs in a variety of formats, e.g.: —interleaving 16-bit data into two 8-bit PROMs —segmenting long programs so that resulting program segments fit into successive PROMs	iPPS FORMAT command allows you to format data in a variety of ways so that it can be loaded into PROMs in various sequences (including interleaving and segmenting).
Verify the accuracy of copying.	iPPS software automatically checks the accuracy of copying.
Compare programming buffer with PROM contents	iPPS VERIFY command compares buffer data with PROM data.
Control the security feature of advanced microcontrollers for unauthorized access.	iPPS KEYLOCK command locks advanced microcontrollers.
Automate routine PROM programming tasks.	ISIS SUBMIT files permit you to store frequently used command sequences. The files can then be activated with a single command.

APPENDIX A
PROM PROGRAMMING EXAMPLES

Displaying (P)ROM contents and programming PROMs are easy tasks with the iPDS system. The following four examples show typical uses of the iPDS system's PROM programming capabilities:

- Examining the contents of a masked ROM
- Duplicating a PROM
- Interleaving a file between two PROMs
- Locking a microcontroller

EXAMPLES

The examples assume that the iPDS system is under control of the iPPS-PDS software. The boldface characters shown on the iPDS screen displays indicate user entries. The key-in sequence below each screen display gives the actual entries that you must key in to obtain the screen display.

Examining the Contents of a Masked ROM

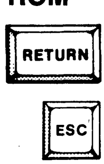
The DISPLAY command lets you examine the contents of a PROM or a masked ROM.

```
PPS> DISPLAY PROM
000000: C3 40 00 20 20 44 20 2D 20 44 49 53 4B 00 20 20  .∂. D - DISK.
000010: 47 20 2D 20 47 45 4E 45 52 41 4C 00 20 20 4B 20  G - GNENERAL.  K
000020: 2D 20 4B 45 59 42 4F 41 52 44 2F 43 52 54 00 FF  - KEYBOARD/CRT..
000030: FF FF FF FF FF FF FF FF C3 36 1C FF FF FF FF FF  ..........6......
000040: F3 DB 80 E6 20 CA 03 08 3E 00 D3 D1 DB 80 E6 01  .... ...>.......
000050: C2 66 00 3E 4F D3 D0 3E 58 D3 D0 3E 89 D3 D0 3E  .f.>O..>X..>...>
000060: 99 D3 D0 C3 76 00 3E 4F D3 D0 3E 98 D3 D0 3E 8A  ....v.>O..>...>.
000070: D3 D0 3E 9C D3 D0 21 00 00 11 00 08 AF 47 78 B2  ..>.../......G{.
000080: CA 8A 00 78 86 23 1B C3 7D 00 78 FE 55 C2 8D 00  ...x.#..}.x.U...
000090: 3E 34 D3 E3 3E 1F D3 E0 3E 00 D3 E0 01 30 00 DB  >4..>...>....0..
0000A0: 80 E6 01 C2 A9 00 01 2C 00 3E 72 D3 E3 79 D3 E1  ........,.>r..y..
0000B0: 78 D3 E1 3E B2 D3 E3 3E 00 D3 E2 3E 16 D3 E2 D3  x..>...>...>....
0000C0: 10 3E 22 D3 60 D3 50 D8 80 E6 04 CA C7 00 DB 80  .>''.,.P.........
0000D0: E6 04 C2 CE 00 AF D3 F0 D3 F0 D3 F0 D3 F1 3E A1  ..............>.
0000E0: D3 F8 3E 23 D3 60 3E C8 D3 E2 3E 00 D3 E2 D3 50  ..>#.,>...>....P
0000F0: 21 EF 00 2B 7C B5 C2 F3 00 DB 80 E6 04 C2 FD 00  |..+|............
000100: 3E 00 D3 E2 3E 16 D3 E2 D3 50 DB 80 E6 04 CA 0A  >...>....P......
000110: 10 DB 80 E6 04 C2 11 01 3E 22 D3 60 D3 50 DB 80  ........>".,.P..
000120: E6 04 CA 1E 01 DB 80 E6 04 C2 25 01 21 00 40 11  ..........%.'.∂.
ENTER <CR> TO CONTINUE $
ABORTED
PPS>
```

280015-3

Key-In Sequence

DISPLAY PROM



280015-4

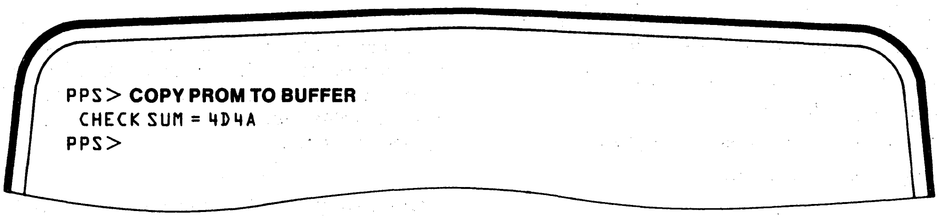
Comments

This example shows the data in the PROM in hexadecimal format, which is the default base in this example. Press the ESC key at any time to end the display. The "S" sign is the echo of the ESC key. You can also display the data in other number bases. Note the ASCII code displayed in the far right column.

Duplicating a PROM

One frequently used application of iPDS PROM programming is copying data from a PROM into a buffer or file, then copying it into another PROM. You can perform this operation using the iPPS-PDS buffer (or an iPDS file for intermediate storage) and the iPPS-PDS COPY commands.

The following example illustrates a direct PROM-to-buffer-to-PROM duplication. If you wish to perform these examples, place the PROM in the PROM socket and reset the iPPS-PDS (using the TYPE command for your type of PROM). A 2716 EPROM that contains sample code is used in this example.



280015-5

Key-In Sequence

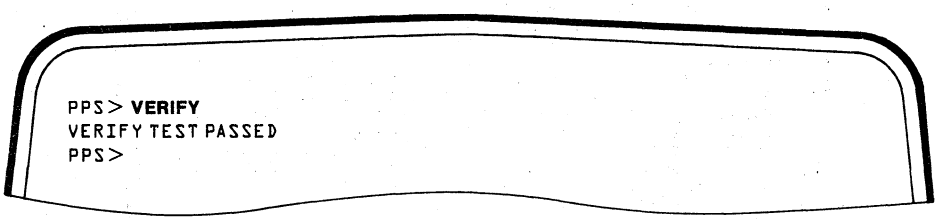
COPY PROM TO BUFFER

280015-6

Comments

This command copies every memory location in the PROM to the buffer beginning at destination address 00H in the buffer. The check-sum is the 2's complement of the 16-bit sum of all the bytes read.

If you want to check the buffer to be sure the data now there matches the original data in the PROM, one command is all that is needed. Enter the VERIFY command, and if the buffer and PROM data match, you will be informed VERIFY TEST PASSED.



280015-7

Key-In Sequence

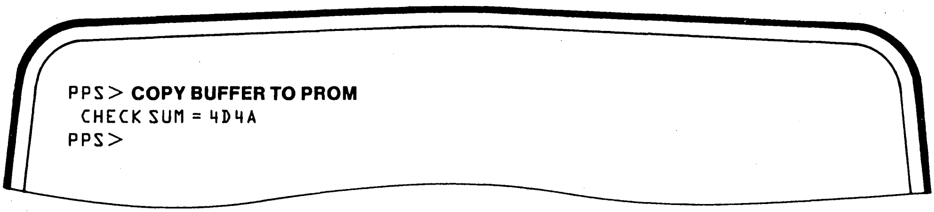
VERIFY

280015-6

Comments

The data in the buffer matches the data in the PROM.

Now that you have verified that the data in the buffer matches the data in the PROM, you are ready to copy the buffer to a blank PROM. Remove the master PROM from the PROM socket and insert the blank PROM. Then use COPY again to copy the contents of the iPPS-PDS buffer to the blank PROM.



280015-8

Key-In Sequence

COPY BUFFER TO PROM

280015-6

Comments

The display of the check-sum and the return of the iPPS prompt indicate that the PROM was successfully programmed.

Note that for copying from the buffer to a PROM, you do not need to use the VERIFY command. The iPPS-PDS software automatically verifies the copying when you copy in this direction.

Interleaving a File between Two PROMS

It is often desirable to have code or data arranged in 16-bit words and stored on a pair of 8-bit PROMs. This is the case, for example, when working with an 8086 microprocessor that reads from and writes to memory on a 16-bit data bus. The data is interleaved between two PROMs, the odd (or low) bytes stored in one PROM and the even (or high) bytes stored in the other PROM. The FORMAT command handles this interleaving automatically.

In the following example, a file written in Intel 8086 hexadecimal format is interleaved into two PROM devices.

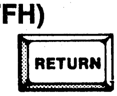
```
PPS>FORMAT DOUBLE.BYT (0,FFFH)
LOGICAL UNIT (BIT=1,NIBBLE=2,BYTE=3,N-BYTE=4)
LU = 3
INPUT BLOCK SIZE (N BYTES)
N = 2
OUTPUT BLOCK SIZE (N BYTES)
N = 1
INPUT BLOCK STRUCTURE:
NUMBER OF INPUT LOGICAL UNITS = 002

LSB
---------
| 00 | 01 |
---------
NUMBER OF OUTPUT LOGICAL UNITS = 001
OUTPUT SPECIFICATION (<CR> TO EXIT):
*
```

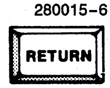
280015-9

Key-In Sequence

FORMAT DOUBLE. BYT (0,FFFH)



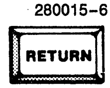
280015-6

3 

280015-6

2 

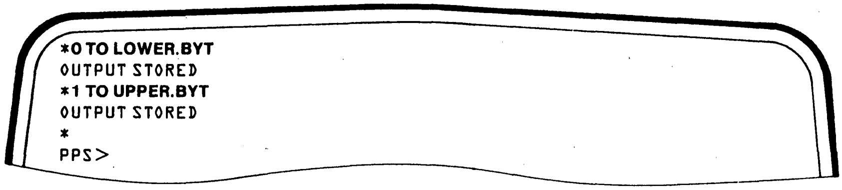
280015-6

1 

280015-6

Comments

In this example, a file called DOUBLE.BYT is split into two files, with alternate bytes being loaded into alternate files. After establishing the FORMAT command and the file name with the first entry, the iPPS software prompts for the size of the logical unit that is going to be manipulated. Byte is selected as the logical unit. You are then prompted to set up the input block size (in this case two bytes) and the output block size (one byte). A diagram of the input block is displayed with the logical units labeled. The least significant bit in the input block is displayed with the logical units labeled. The least significant bit in the input block is shown on the left. The number of logical units in the output block is also displayed. You are then prompted with an asterisk (*) to enter the output specification.



280015-10

Key-In Sequence | **Comments**

0 TO LOWER.BYT

280015-6

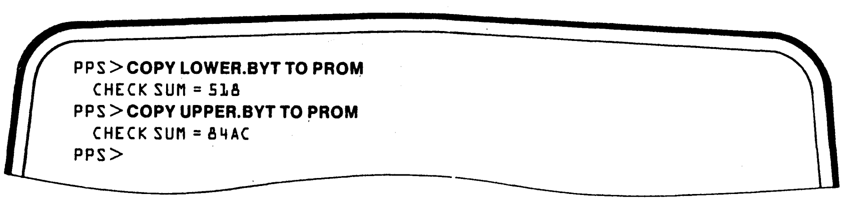
1 TO UPPER.BYT

280015-6

280015-6

Once the size of the logical unit, the input block size and the output block sizes have been established, you are prompted for the output specification (how you want the data in the file to be manipulated in terms of logical units). This example specified that the least significant byte in each input block be stored in a file titled LOWER.BYT in the default drive. The iPPS software then sorts through the DOUBLE-BYT file. Next it specifies that the most significant byte be stored in a file titled UPPER.BYT. The iPPS software then sorts through the DOUBLE-BYT file and copies every odd byte to the UPPER-BYT file. OUTPUT STORED is displayed after each output specification is implemented. You then have the option of entering another output specification. Pressing RETURN exits the FORMAT command and returns the iPPS prompt.

You can use the two files created with this FORMAT operation to program two PROMs, which you can then install in parallel to provide 16-bit data words to a 16-bit microprocessor. To copy the files the PROMs, use the COPY command as follows.



280015-11

Key-In Sequence | **Comments**

COPY LOWER.BYT TO PROM

280015-6

COPY UPPER.BYT TO PROM

280015-6

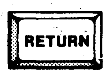
You must install a blank PROM in the personality module before entering each COPY command.

Locking a Microcontroller

After programming a microcontroller, you can protect it from unauthorized access by locking it with the KEYLOCK command (the KEYLOCK command cannot be used with all EPROMs). The following example locks an 8751H microcontroller, which then cannot be unlocked without erasing it.

```
PPS>KEYLOCK
EXECUTE--Y/N? Y
PPS>
```

280015–12

Key-In Sequence		Comments
KEYLOCK	 280015–6	Entering Y locks the EPROM. If you enter N, the command terminates and EPROM remains unlocked.
Y	 280015–6	